Machine Learning Engineer Nanodegree

Capstone Proposal

Francisco Ramos
May 31st, 2019

Proposal

Domain Background

With the boom of AI the financial industry is experiencing interesting changes and improvements. Many companies are starting to integrate Machine Learning in their businesses. Fraud prevention, risk management, portfolio management, investment prediction, process automation, are some of applications of ML in finance. The Stock Market and Algorithmic Trading are other areas benefiting from it. Traders are embracing some of the new techniques and algorithms to improve their performance.

Time series forecasting has become a hot topic with the advances in Machine Learning. Time series analysis and predictions is one of the oldest and most applied data science techniques in finance, but new ones are emerging, more sophisticated and powerful, providing with more accuracy, using the power of Deep Learning.

The Stock Market has always been an area of interest for me. I always felt fascinated by the “science” of time series forecasting. It’s certainly very intriguing how stock prices behave, initially quite randomly, but put together enough historical data, the right indicators and the latest ML models and there is no doubt that we could actually predict stock movements with high accuracy, beating current models such as ARIMA.

Problem Statement

Stock Market prediction is a difficult task. There are many variables that affect how the share prices change over time. Combination of different factors make them volatile and therefore quite unpredictable, or at least hard to predict with high accuracy.

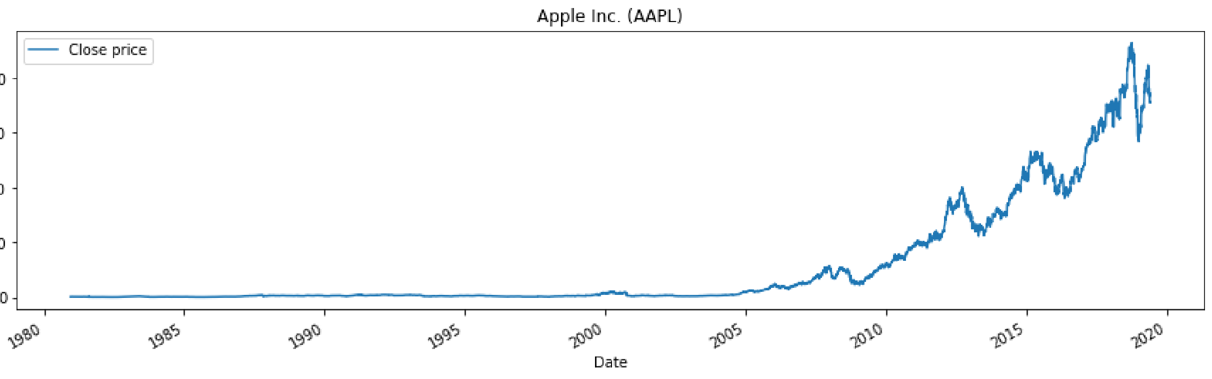
According to financial theory, the stock market prices evolve following the so called random walk, but there are enough evidences that dismiss this theory, and they can be predicted based on historical information.

Stock Market analysis is divided into two parts: Fundamental Analysis and Technical Analysis. Fundamental Analysis involves analyzing the company's future profitability on the basis of its current business environment and financial performance. Technical Analysis, on the other hand, includes reading the charts and using statistical figures to identify the trends in the stock market. In this project I'm gonna focus on the technical analysis part.

There a few solutions out there for time series forecasting such as ARIMA models (already mentioned), Support Vector Machines, or state-of-the-art LSTM models

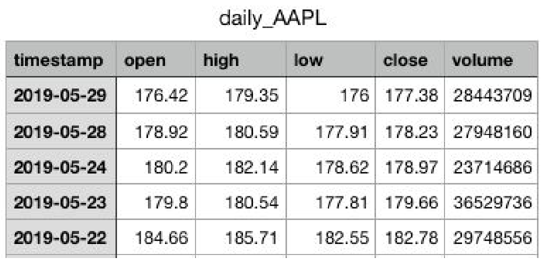
Datasets and Inputs

For this project I'm gonna make use of historical stock market data that one can easily obtain for free from Kaggle, Yahoo! Finance or Alpha Vantage (API key is required). I'm gonna focus on Apple stock prices just because, well, it's a well known company and I'm curious to know what I can get out of it.



I'll make use of S&P 500 Index as an indicator feature, because it represents a good segment of the financial market in US. It's based on the market capitalizations of 500 large companies – Apple included – which could indicate change and tendency in the future, but my financial knowledge is very limited so this is just my guess.

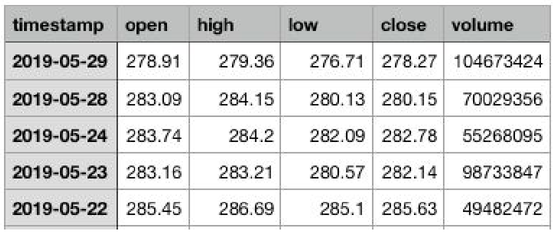
Our dataset is the typical daily Open-High-Low-Close prices with Volume, which we're gonna use as features, besides others I'll mention later.


daily_AAPL

timestamp	open	high	low	close	volume
2019-05-29	176.42	179.35	176	177.38	28443709
2019-05-28	178.92	180.59	177.91	178.23	27948160
2019-05-24	180.2	182.14	178.62	178.97	23714686
2019-05-23	179.8	180.54	177.81	179.66	36529736
2019-05-22	184.66	185.71	182.55	182.78	29748556

S&P Index follows same format:

daily_SPY



timestamp	open	high	low	close	volume
2019-05-29	278.91	279.36	276.71	278.27	104673424
2019-05-28	283.09	284.15	280.13	280.15	70029356
2019-05-24	283.74	284.2	282.09	282.78	55268095
2019-05-23	283.16	283.21	280.57	282.14	98733847
2019-05-22	285.45	286.69	285.1	285.63	49482472

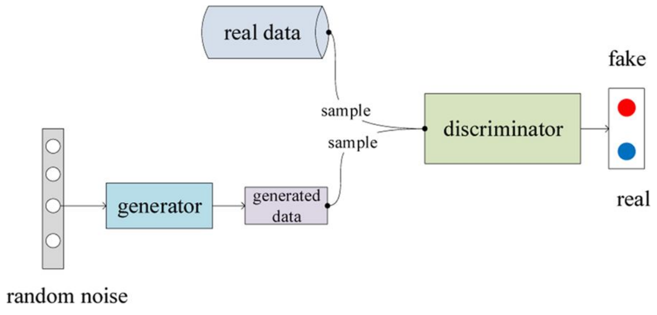
Solution Statement

As mentioned earlier, Stock Market prediction is an important issue in the financial world. It's also an old one and many different models and solutions have been designed in order to predict stock movements. Today experts and researchers are making use of the latest advances in Deep Learning, and they look quite promising. RNN using LSTM or Deep Reinforcement Learning are being explored, but a few years ago, Ian Goodfellow and other researchers came up with an interesting framework that's able to mimic any distribution of data. The name, Generative Adversarial Nets (GANs). There has been interesting applications using them such as generating very realistic images, and the results are stunning.

The question arises: can we also use this powerful framework to generate "realistic" time series? And if so, can we actually use it to generate future stock prices movements, and therefore predict stock prices with some high level of accuracy?. After some research, I've noticed that there is no as many attempts to use GANs for time series forecasting as the other two solutions mentioned above, using LSTM or RL. So I've decided to present, for this project, my own solution using Generative Adversarial Nets.

– As a side note, as I explored the idea, I realized that, if GANs are able to generate very realistic time series based on an original one, learning the distribution behind it and picking up patterns, this could also be used to replace Monte Carlo simulations. Not only that, but it could even outperform it, which would improve financial areas such as risk management. But this is a subject for another project.

I'll try to give a high-level overview of how GANs work in order to understand how they could be used to predict stock price movements. A GAN network consists of two models, a Generator (G) and Discriminator (D). G uses random data to generate data indistinguishable of, or extremely close to, the real data. Its purpose is to learn the distribution of the real data. Randomly, real or generated data, is fitted into D, which acts as a classifier and tries to understand whether the data is coming from G or is the real data. D estimates the probabilities of the incoming sample to the real dataset. Then, the losses from G and D are combined and propagated back through G. G's loss depends on both G and D. This is the step that helps G learn about the real data distribution. If G doesn't do a good job at generating realistic data (having the same distribution), it'll be very easy for D to distinguish generated from real data. Therefore D's loss will be very small and G's loss will be big. The process goes on until D is no longer able to distinguish generated from real data, since G has learned very well the distribution behind the real data and it's able to produce very realistic one. There are other, in more detail, explanations of how GANs work, but it's not the purpose of this proposal to go so much into detail.



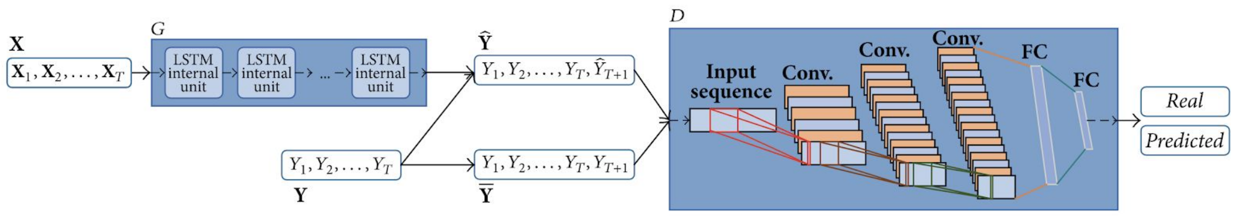
Architecture of a GAN

In order for our Generator to predict future prices we're gonna make use of a type of Recurrent Neural Network, LSTM network. RNN are used in time-series data because they're able to keep track of previous data and capture patterns that can help to predict the output of future time steps. RNN suffer from vanishing gradient. LSTM networks keep the gradients from vanishing.

For our Discriminator we're gonna make use of Convolutional Neural Networks. CNNs are normally used in Computer Vision. They're excellent at extracting features from features. In our case, data points form trends. Trends in turn form patterns. They're also powerful on spatial data. It'll help to understand data points closer to each other and points far apart from each other. Days closer to each other are more related to one another.

So what's the whole idea?, We have a sequence of days, D = {d_1, d_2, …, d_n}. Each day has features, X = {X_1, X_2, …, X_n}, plus closing prices Y = {Y_1, Y_2, …, Y_n}. The goal is to predict the closing price Y_{n+1} for the next D_{n+1}. We can decide then to generate more days, months or years in the future. We could also try to predict price direction, but this could add more complexity to the project, so it's still yet to be decided.

At the moment of writing this proposal, I haven't worked out yet the architecture I'm gonna used in terms of amount of layers, units, regularization methods, hyperparameters, etc… but it'll look more or less like the image below:



Where $\bar{Y}$ is the real sequence and $\hat{Y}$ is the predicted one. This image has been taken from one of the very few resources found about the subject.

Another important point to mention is the need to perform feature engineering since we only have 4 features, Open, High, Low, and Volume the discriminator can use to learn from and decide whether the generated sequence is real or fake. At the beginning I mentioned that I'm gonna be using also S&P 500 daily data as new features. We can also include different rolling windows of Moving Average, Bollinger Bands and Exponential Moving Average. I could make use of this interesting package to add different indicators to my dataset, such as momentum indicators, volume indicators, volatility indicators, etc.

Benchmark Model

I mentioned earlier on current methods used for time series forecasting, such as ARIMA models and more recent LSTM networks. Would be interesting to use them both to evaluate the performance of the proposed method comparing which one is closer to the ground truth. I'm gonna focus on predicting the next D days closing prices, so we can use different metrics to calculate the error and compare these results. Also plotting the predicted time series would be a great way to visualize the different predictions.

Evaluation Metrics

A good metric to use would be Root Mean Square Error, RMSE, because it reflects the stability between the original data and generated data:

$$RMSE = \sqrt{\frac{1}{N}\sum_{n=1}^{N}\left(x_{[n]} - \widehat{x_{[n]}}\right)^2}.$$

where $x_{[n]}$ is the nth real point, $x_{[n]}$-hat is the nth generated point, and N is the length of the generated sequence.

I could also use the Percent Root Mean Square Difference, PRD, which is a widely used distortion measurement method:

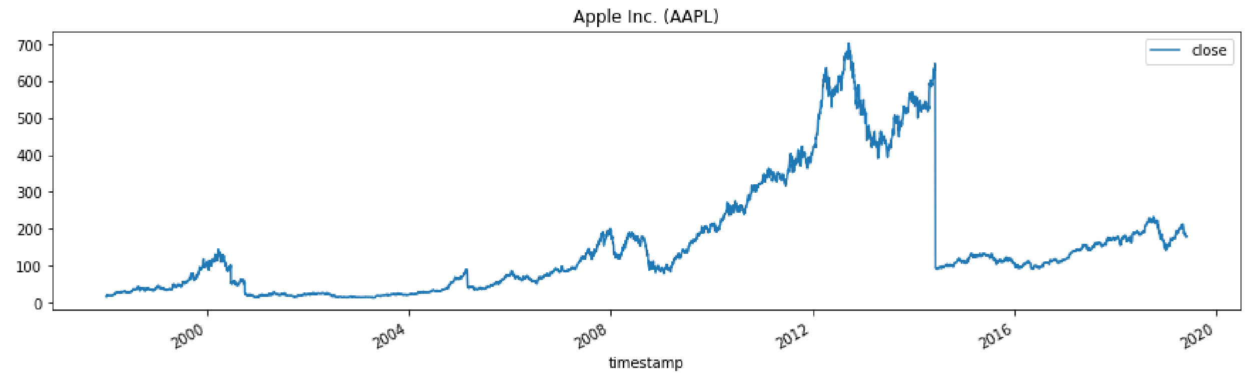
$$PRD = \sqrt{\frac{\sum_{n=1}^{N}\left(x_{[n]} - \widehat{x_{[n]}}\right)^2}{\sum_{n=1}^{N}\left(x_{[n]}\right)^2} \times 100},$$

After some research I discovered Fréchet Distance, FD, which is a measure of similarity between curves that takes into consideration the location and ordering of points along the curves, especially in the case of time series data. But I still don't understand well the formula, so I'll keep it in mind and do more research to understand how to use it if there is time for it.

Project Design

The following is a summary of the potential workflow to approach the solution of using GANs to predict Stock Market prices:

1. I'll start off with a little bit of data exploration. While visualizing Apple stock prices I noticed some strange, a sudden drop in June 9th, 2014.

   What happened here was that there was a Stock Split. No reason to go into details about this, but basically this could cause very strange results, so the solution is to divide by 7 all the prices before the split.

2. Next step would be to add more features to our dataset. I'm not an expert in finance, so I might get some of these features wrong. As mentioned in the Datasets and Inputs section, I'll be using also S&P 500 index, and add its OHLC and Volume as new features. Also considering the idea of using, or replacing S&P 500 with daily volatility index, VIX. I could be also adding other assets that might somehow influence Apple stock prices. But that needs some more research.

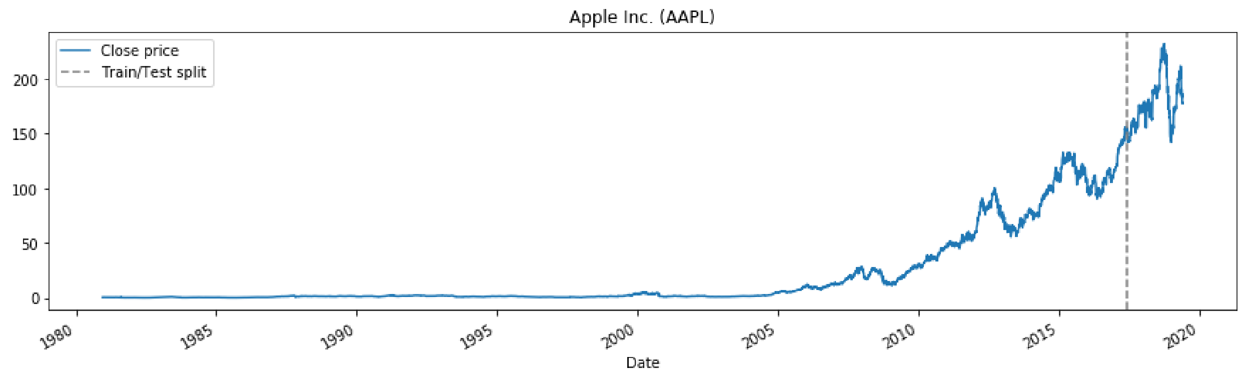
3. Following I'm gonna add a few technical indicators to the dataset such as different windows of Moving Average, MACD, Bollinger Bands, Exponential Moving Average and Momentum. This has been strongly inspired by this work, along with Fourier transforms for trend analysis. There are many others to be considered such as the ones suggested by this python lib.

```python
def add_technical_indicators(dataset):
    # Create 7 and 21 days Moving Average
    dataset['ma7'] = dataset['price'].rolling(window=7).mean()
    dataset['ma21'] = dataset['price'].rolling(window=21).mean()

    # Create MACD
    dataset['26ema'] = pd.ewma(dataset['price'], span=26)
    dataset['12ema'] = pd.ewma(dataset['price'], span=12)
    dataset['MACD'] = (dataset['12ema']-dataset['26ema'])

    # Create Bollinger Bands
    dataset['20sd'] = pd.stats.moments.rolling_std(dataset['price'],20)
    dataset['upper_band'] = dataset['ma21'] + (dataset['20sd']*2)
    dataset['lower_band'] = dataset['ma21'] - (dataset['20sd']*2)

    # Create Exponential moving average
    dataset['ema'] = dataset['price'].ewm(com=0.5).mean()

    # Create Momentum
    dataset['momentum'] = dataset['price']-1

    return dataset
```

4. After adding all these features, we might want to find out whether all of them are really indicative of direction Apple stock will take. We can use XGBoost to see how useful these features are.

5. Normalization of these features would be the next step, using `sklearn.preprocessing.MinMaxScaler()`, to help with the training. This article points out something quite interesting to keep in mind: *"Different time periods of data have different value ranges, you normalize the data by splitting the full series into windows. If you don't do this, the earlier data will be close to 0 and will not add much value to the learning process. Here you choose a window size of 2500"*. This is something I need to investigate during training.

6. At this point we need to decide how we're gonna split our set into train and test. An example would be: all data points until last two years for training; data points of last two years for testing:



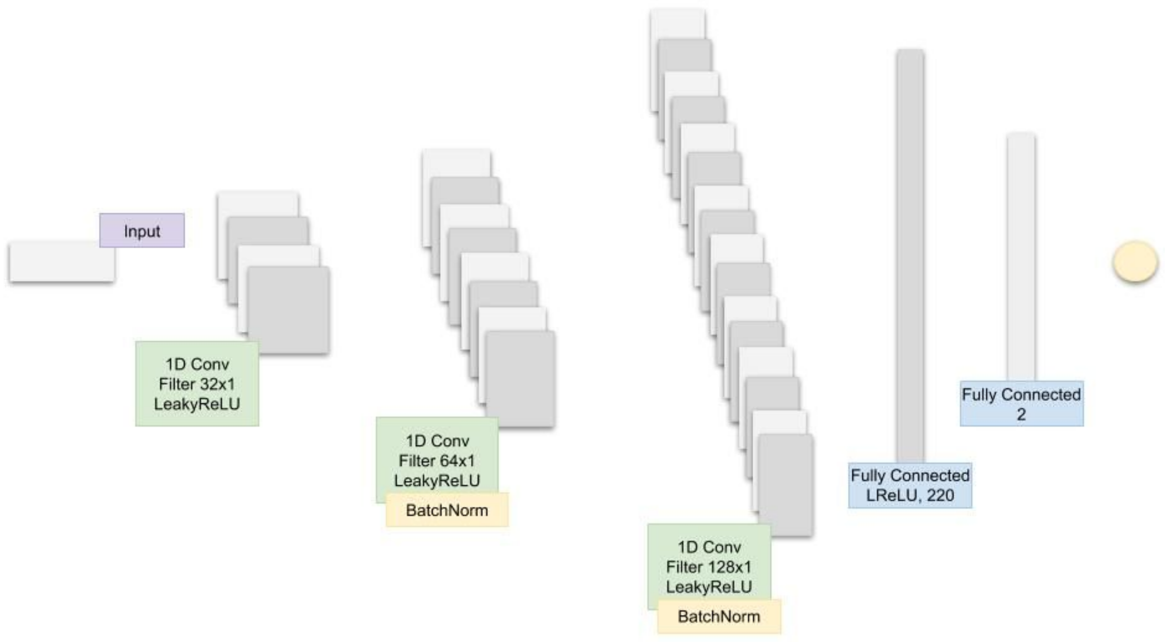
Note: I might change the year of split depending on results.

7. As discussed in the Benchmark Model section, I'd like to compare my solution with other ones, such as ARIMA, which is a old but very effective method for time series forecasting, and one of the latest and quite promising one, using RNN-LSTM model. I found a few implementations in Python of ARIMA model that I can use and adapt to this problem.

8. Following ARIMA implementation, forecasting and evaluation, I'll move on to LSTM model and repeat the process. The architecture used for this model is still to be decided, but I might start with something as simple as the code below and then play around with other layers and neurons, regularization, batch normalization, etc:

```
# define model
model = Sequential()
model.add(LSTM(units=50, activation='relu', input_shape=(n_steps, n_features)))
model.add(Dropout(0.2))
model.add(LSTM(units=50))
model.add(Dropout(0.2))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')
# fit model
model.fit(X, y, epochs=200, verbose=0)
```

9. Finally our GAN implementation. As mentioned, I'll be using LSTM for our Generator, and in fact I'll be using the same architecture as the LSTM model used as benchmark, and CNN for our Discriminator. I'm gonna try the following architecture for the Discriminator:



Now, it's important to mention here that training a GAN is quite difficult. Models may never converge and mode collapses are common. There are lots of articles and references about this subject. GANs are quite new to me, so this part will be the one that'll take me the longest. It's hard for me to decide at this point which variant of GAN I'll be using. Wasserstein GAN are quite promising because they seem to address many problems during training. There are different loss functions that I still need to understand so I'll spend long time in this step experimenting.

10. In this last step I'll be comparing results using the metrics mentioned earlier on, RMSE, PDR and, perhaps, Fréchet distance, and plotting the predictions for a better visualization of this comparison.